OUKA
Osaka University Knowledge Archive

Title	Fast Density-Peaks Clustering: Multicore-based Parallelization Approach
Author(s)	Amagata, Daichi; Hara, Takahiro
Citation	Proceedings of the ACM SIGMOD International Conference on Management of Data. 2021, p. 49-61
Version Type	AM
URL	https://hdl.handle.net/11094/92848
rights	© 2021 ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in Amagata D., Hara T.. Fast Density-Peaks Clustering: Multicore-based Parallelization Approach. Proceedings of the ACM SIGMOD International Conference on Management of Data , 49 (2021); https://doi.org/10.1145/3448016.3452781.
Note	

Osaka University Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

Osaka University

Fast Density-Peaks Clustering: Multicore-based Parallelization Approach

Daichi Amagata
Osaka University, PRESTO
Japan
amagata.daichi@ist.osaka-u.ac.jp

Takahiro Hara
Osaka University
Japan
hara@ist.osaka-u.ac.jp

ABSTRACT

Clustering multi-dimensional points is a fundamental task in many fields, and density-based clustering supports many applications as it can discover clusters of arbitrary shapes. This paper addresses the problem of Density-Peaks Clustering (DPC), a recently proposed density-based clustering framework. Although DPC already has many applications, its straightforward implementation incurs a quadratic time computation to the number of points in a given dataset, thereby does not scale to large datasets.

To enable DPC on large datasets, we propose efficient algorithms for DPC. Specifically, we propose an exact algorithm, Ex-DPC, and two approximation algorithms, Approx-DPC and S-Approx-DPC. Under a reasonable assumption about a DPC parameter, our algorithms are sub-quadratic, i.e., break the quadratic barrier. Besides, Approx-DPC does not require any additional parameters and can return the same cluster centers as those of Ex-DPC, rendering an accurate clustering result. S-Approx-DPC requires an approximation parameter but can speed up its computational efficiency. We further present that their efficiencies can be accelerated by leveraging multicore processing. We conduct extensive experiments using synthetic and real datasets, and our experimental results demonstrate that our algorithms are efficient, scalable, and accurate.

1 INTRODUCTION

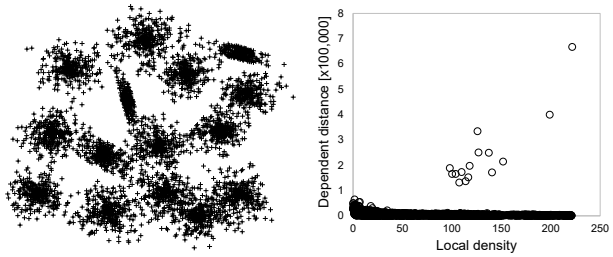
Given a set P of n points in a d-dimensional space, clustering them aims at dividing P into some subsets, i.e., clusters. This multi-dimensional point clustering is a fundamental task for many data mining applications. Density-based clustering particularly supports them well because it can discover clusters of arbitrary shapes. This is an advantage over other clustering such as k-means [17, 42].

DPC Framework. Recently, as a novel density-based clustering framework, *Density-Peaks Clustering* (DPC) has been proposed in [30], and we focus on DPC in this paper. Given a point set P, DPC computes, for each point $p_i \in P$,

- *local density* ρ_i: the number of points p_j whose distances between p_i and p_j are less than d_{cut}, which is a user-specified cutoff parameter, and
- *dependent distance* δ_i: the distance from p_i to its nearest neighbor point in P with higher local density than ρ_i (i.e., *dependent point*).

Then DPC identifies

- *noises*: points with less local density than ρ_{min}, and
- *cluster centers*: non-noise points, whose dependent distances are at least δ_{min} (each cluster center should have a comparatively long dependent distance, as its local density is *peak* at its area).



(a) Dataset S2 (b) Decision graph of S2

Figure 1: Illustration of S2 and its decision graph

After that, the remaining points are assigned to the same cluster as their dependent points.

Advantages of DPC. In addition to the inherent advantages of density-based clustering, DPC has some advantages. One of them is that, even if users (or applications) are not domain experts, they can intuitively select cluster centers and noises from a *decision graph*, which visualizes $\langle \rho_i, \delta_i \rangle$ into a 2-dimensional space.

Example 1. *Figure 1 illustrates (a) dataset S2 [16] and (b) its decision graph. S2 has 15 Gaussian clusters. The decision graph depicts that 15 points have comparatively large dependent distances. That is, it visually suggests that there are 15 clusters in the dataset and does a threshold (δ_{min}) for obtaining their centers (recall that cluster centers tend to have comparatively long dependent distances). Thanks to this observation, users can easily specify δ_{min} and ρ_{min}.*

Another advantage of DPC is that it can divide a dense space into some sub-spaces if the space has some density-peaks. On the other hand, a famous density-based clustering DBSCAN [14] (and its variants) cannot deal with this case well, for example:

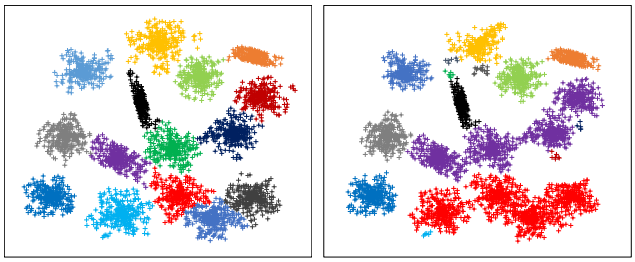
Example 2. *Figure 2 illustrates the clustering results of DPC and DBSCAN on S2 (best viewed in color). The parameters of DBSCAN are specified so that 15 clusters are obtained from OPTICS [4]. In this figure, DBSCAN fails to obtain the correct clusters (some clusters are merged to their neighbor clusters), whereas DPC does not.*

This example presents that DPC is robust to data distributions, and [7, 42] have also confirmed this observation (on other datasets).

Motivation. Real-life applications often generate datasets with arbitrary shaped clusters that may not be clearly separated [12] (e.g., they may have points existing between close clusters), and DPC deals with them well even if they have such clusters. Besides this, DPC supports easy selection of cluster centers and noises (see

Table 1: The time complexity of each algorithm on a single thread for fixed dimensionality d (under a reasonable assumption). Note that n is the cardinality of a given dataset, M is the number of compound LSHs, b is the average bucket size of an LSH, and ρ_{avg} is the average local density.

Algorithm	Local density computation	Dependent distance computation	Total
Scan	$O(n^2)$	$O(n^2)$	$O(n^2)$
LSH-DDP [42]	$O(M \sum b^2)$	$O(M \sum b^2)$	$O(M \sum b^2)$
CFSFDP-A [7]	$O(n^2)$	$\Omega(n^2)$	$\Omega(n^2)$
Our algorithms in this paper	$O(n(n^{1-1/d} + \rho_{avg}))$	$O(n^{2-1/d})$	$O(n(n^{1-1/d} + \rho_{avg}))$



(a) DPC (b) DBSCAN

Figure 2: Difference of the clustering results (clustering quality comparison) between DPC and DBSCAN on S2

Example 1). DPC therefore has been already employed in many fields, e.g., neuroscience [27], medical application [34], document summarization [43], graphics [24], and computer vision [36]. This fact observes the importance of DPC.

The above applications obviously require fast DPC algorithms to deal with a large number of points. However, the computational efficiency of DPC has not been tackled much so far.

Suffering from the quadratic computation time. A straightforward computation of the local density and dependent distance (point) for a point $p_i \in P$ is to scan the whole P, which incurs $O(n)$ time. Therefore, computing the local density and dependent distance for all points in P incurs $O(n^2)$ time. Some existing works [7, 42] reduced the practical computation time, but they still suffer from the quadratic computation time.

Non parallel-friendly. Leveraging parallel processing environments is a practical approach to scale well to large datasets. Nowadays many CPU cores (or threads) are available in a single machine, so multicore-based parallel algorithms for many database operations have been considered [15, 29]. As clustering is an important database and data mining operation that needs to deal with large datasets, a parallel-friendly DPC algorithm is motivated. Although there exist parallel DPC algorithms [42], they do not consider load balancing, which limits the improvement of efficiency.

Contributions. Motivated by the above facts, we present parallel DPC algorithms that run in $o(n^2)$ time under a reasonable assumption. We summarize, in Table 1, the worst time complexities of existing and our algorithms, and our main contributions below[1].

- *Ex-DPC* (§3). We first propose Ex-DPC, an exact algorithm that exploits a kd-tree [8]. This algorithm computes the local densities and dependent distances of all points in $O(n(n^{1-1/d}+\rho_{avg}))$ time and $O(n^{2-1/d})$ time, respectively.
- *Approx-DPC* (§4). Second, we propose Approx-DPC, an approximation algorithm that relaxes the constraint of dependent points *by approximating dependent points*. This algorithm improves the performance of local density computation and computes approximate dependent points for many points in $O(1)$ time. Although the worst time complexity of Approx-DPC is the same as that of Ex-DPC, its time is shown to be faster than $O(n(n^{1-1/d}+\rho_{avg}))$ under a practical assumption.
- *S-Approx-DPC* (§5). Our last algorithm, S-Approx-DPC, is also an approximation algorithm. S-Approx-DPC employs the concepts of *grid sampling* and *cell-based clustering* so that users can improve computation time through an approximation parameter. We theoretically discuss the efficiency of S-Approx-DPC.
- *Parallelization of our algorithms.* We also present how to parallelize our algorithms in a multicore environment.
- *Empirical evaluation* (§6). We conduct experiments using synthetic and real datasets. Our experimental results demonstrate that Ex-DPC outperforms the state-of-the-art exact algorithm significantly. Also, our approximation algorithms beat the state-of-the-art approximation algorithms and are usually more than 10 times faster than them.

We formally define the problem in this paper and introduce related works in §2, and this paper is concluded in §7.

2 PRELIMINARY

2.1 Problem Definition

Let P be a set of n points in a d-dimensional space $\mathbb{R}^d$. Density-Peaks Clustering (DPC) aims at dividing P into some subsets based on density-peaks. To this end, DPC requires two important metrics, *local density* and *dependent distance*. We formally define them.

Definition 1 (Local density). *Given a point $p_i \in P$ and a cutoff distance d_{cut}, its local density ρ_i is:*

$$\rho_i = |\{p_j \mid dist(p_i, p_j) < d_{cut}, p_j \in P\}|. \quad (1)$$

That is, the local density of $p_i \in P$ is the number of points p_j such that $dist(p_i, p_j)$ is less than a user-specified threshold d_{cut}[2].

[1] This is an error-corrected version of [3]. Note that the algorithms do not change, so their practical performances also keep efficient.

[2] The analyses in [3] are essentially valid iff the local density is defined as an axis-parallel rectangle. This paper adds some conditions to make the analyses still valid for the circular (or hyper-ball) case.

Definition 2 (Dependent point). *Given a point $p_i \in P$, its dependent point q_i satisfies:*

$$q_i = \arg\min_{\rho_i < \rho_j} dist(p_i, p_j).$$

That is, the dependent point of p_i is the nearest neighbor point to p_i among a set of points with higher local densities than ρ_i (ties are broken arbitrarily).

Definition 3 (Dependent distance). *Given a point p_i, its dependent distance δ_i is $dist(p_i, q_i)$.*

The point with the highest local density in P cannot have a dependent point, so its dependent distance is simply ∞.

Next, we define *noise* and *cluster center*, which are respectively based on local density and dependent distance.

Definition 4 (Noise). *If a point $p_i \in P$ has $\rho_i < \rho_{min}$, p_i is a noise.*

Definition 5 (Cluster center). *If a non-noise point $p_i \in P$ has $\delta_i \geq \delta_{min}$, where $\delta_{min} > d_{cut}$ is a user-specified threshold, p_i is a cluster center.*

We can specify ρ_{min} and δ_{min} at the same time when d_{cut} is specified or after a decision graph is viewed. Note that ρ_{min} is specified to remove points with (very) small local densities (e.g., $\rho_{min} = 10$). Besides, δ_{min} is specified so that points with much longer dependent distances than the other points (like ones in Figure 1(b)) are selected as cluster centers.

Once a cluster center, say p_i, is identified, we consider that the dependent point of p_i is itself. There are points p_j whose dependent points are p_i. Also, there are points p_k whose dependent points are p_j. From this observation, we say that p_k (and also p_i and p_j) is (are) *reachable* from the cluster center p_i. Based on this, we define:

Definition 6 (Cluster). *Given P, a cluster C, whose cluster center is p_i, is a non-empty subset of P such that non-noise points $p \in C$ are reachable from p_i.*

Since each point $p \in P$ has a single dependent point, p belongs to a single cluster. Therefore, DPC provides a unique set of clusters.

We assume that P is memory-resident and in a low-dimensional Euclidean space [17–20, 32, 37]. For high-dimensional data, dimensionality reduction, e.g., [26], is commonly used to obtain meaningful clusters [1]. Therefore, our assumption fits into practical usage. In addition, we put a practical assumption that d_{cut} is sufficiently small to satisfy $\rho_{avg} \ll n$ (or $\rho_{avg} < n$), where ρ_{avg} is the average local density in P. If $\rho_{avg} \approx n$, we cannot see density-peaks, because most points in P have very large local densities. This case hence renders a bad clustering result. Last, this paper assumes a single machine with a multicore CPU (or with multicore CPUs). The other parallel computation environments are not the scope of this paper.

2.2 Straightforward algorithm

One simple solution of DPC is as follows. Given a set P of points,

(1) For each point $p_i \in P$, we compute its local density ρ_i by a linear scan of P.
(2) We sort P in descending order of local density.
(3) Then, for each point $p_i \in P$, we compute its dependent point q_i through a linear scan of P with early termination (we can stop the scan when we access p_j such that $\rho_i \geq \rho_j$). To efficiently process the next operation, each point p_i in P maintains the identifiers of points whose dependent points are p_i (e.g., if p_i is q_j, p_i maintains j).
(4) Last, we determine noises and cluster centers and then propagate the corresponding cluster label for each non-noise point in the manner of depth-first-search from the cluster centers.

The above operations incur $O(n^2)$, $O(n \log n)$, $O(n^2)$, and $O(n)$ time, respectively. The above algorithm hence incurs $O(n^2)$ time. This is not tolerant of large datasets, thus we devise efficient solutions. The above label propagation operation is already efficient and common to our algorithms.

2.3 Related Work

k-means clustering is to find a set of clusters that minimizes the sum of squared deviations of points in the same cluster. This is well-known to be NP-hard, and many studies developed fast approximation algorithms with a small error [5, 6]. One limitation of k-means clustering is that it normally provides ball-shaped clusters and cannot deal with complex-shape clusters. Another limitation is that k-means clustering is sensitive to noises (or outliers). Some papers addressed the problem of k-means clustering in the presence of noises [9, 23]. However, this problem requires the number of noises in advance, which is not practical.

DBSCAN. In this clustering, each point $p \in P$ is evaluated whether it is a *core point* or not, based on two input parameters ε and $minPts$ (if at least $minPts$ points exist within ε from p, p is a core point) [14]. DBSCAN assumes that, if the distance between two core points is within ε, there is a connection between them. Informally, DBSCAN forms a cluster by connecting core points in the above way.

We do not say that DPC can replace DBSCAN, since an appropriate clustering algorithm for a given dataset is dependent on its data distribution. However, DPC is more effective for datasets that have skewed density and points existing between close clusters. This is because DBSCAN may consider multiple dense point groups as a single cluster if there are points existing in the border spaces between different groups, while DPC is robust to such a data distribution, as shown in Figure 2(b). This is the main difference between DPC and DBSCAN[3]. Variants of DBSCAN, such as OPTICS [4], inherit the above problem [7] (see Example 2) and do not scale to large datasets, because they incur $O(n^2)$ time.

DBSCAN is costly, so efficient stand-alone algorithms [11, 31] and parallel ones [25, 40] have been proposed. Due to the computational hardness of exact DBSCAN [17, 19], approximation algorithms for DBSCAN have been receiving attention [17, 19, 32, 38].

Density-peaks clustering. Many applications have employed DPC, and some variants of DPC [10, 27, 35, 41] and streaming DPC [21, 34] have also been proposed. This paper follows the original definition [30] and considers a static P.

One state-of-the-art parallel and approximation DPC algorithm is LSH-DDP [42]. Although LSH-DDP is originally designed for distributed computing environments (MapReduce), it can work in

[3] A further discussion about their difference can be found in [21]. Besides, [7, 21, 34, 42] compared DPC and DBSCAN w.r.t. clustering qualities, and the results are similar to Figure 2, i.e., clustering results are (totally) different. Interested readers may refer to them.

multicore environments. LSH-DDP employs locality-sensitive hashing (LSH) [13] to partition P into some buckets (i.e., disjoint subsets of P) so that points in the same bucket are similar to each other. For each $p \in P$, LSH-DDP computes an approximate local density of p by scanning the bucket that includes p. Then, an approximate dependent point of p is retrieved from the bucket that includes p. If the distance between p and its approximate dependent point does not seem accurate, LSH-DDP computes its dependent point by scanning P. LSH-DDP can reduce computation time by approximating local densities and dependent distances, but it does not consider load balancing for parallel processing.

One state-of-the-art exact algorithm for DPC is CFSFDP-A [7]. This algorithm selects *pivot points* and utilizes triangle inequality to avoid unnecessary distance computation. In the pre-processing phase, CFSFDP-A employs k-means clustering to select pivot points, that is, k points are selected as pivot and they are the centroids of k clusters. For each point $p_i \in P$, CFSFDP-A computes a candidate set of points p_j such that $dist(p_i, p_j)$ may be less than d_{cut} by utilizing pivots and triangle inequality. Its dependent point computation is done in a similar manner. Recall that k-means clustering is sensitive to noises, so its pivot selection does not provide good filtering power, meaning that the candidate size is still large. An approximation algorithm was also proposed in [7], but it does not consider density-peaks and provides clustering results with low accuracy.

3 EX-DPC

Ex-DPC, our exact DPC algorithm, assumes that a point set P is indexed by an in-memory kd-tree $\mathcal{K}$, which provides efficient tree update and similarity search in practice while retaining theoretical performance guarantees[4]. First, we present Ex-DPC with a single thread. We then explain how to parallelize Ex-DPC.

Local density computation. Definition 1 suggests that computing the local density of a point $p_i \in P$ corresponds to doing a range search whose query point and radius are p_i and d_{cut}, respectively. Because the kd-tree supports efficient range search, we simply utilize it. That is, for each $p_i \in P$, Ex-DPC runs a range search on $\mathcal{K}$ to obtain ρ_i. This approach is simple but has good theoretical performance.

Before we analyze the performance of Ex-DPC, we put the following assumption.

Assumption 1. *For each point $p \in P$, let OUT be the number of points in the bounding rectangle of the circle defined in Equation (1). We assume that $OUT \approx \rho$.*

This is reasonable and practical for small d_{cut}. Hereinafter, our theoretical analyses assume that Assumption 1 holds, i.e., (some of) the subsequent lemmas and theorems hold under Assumption 1.

Lemma 1. *The time complexity of the local density computation in Ex-DPC is $O(n(n^{1-1/d} + \rho_{avg}))$ under Assumption 1.*

Proof. The time complexity of a range search with query point p_i on a kd-tree is $O(n^{1-1/d} + \rho_i)$ [33]. Therefore, the time complexity of the local density computation of Ex-DPC is $O(\sum_P (n^{1-1/d} + \rho_i)) = O(n(n^{1-1/d} + \rho_{avg}))$. □

Dependent point computation. Recall the constraint of dependent points: the dependent point of p_i has to be retrieved from a set of points with higher local densities than ρ_i. Since the local density depends on d_{cut}, it is hard to build a data structure for obtaining dependent points efficiently in a pre-processing phase. Although the kd-tree supports efficient nearest neighbor search, it is not guaranteed that the nearest neighbor point of p_i has higher local density than ρ_i. Hence it is challenging to compute dependent points efficiently. We overcome this challenge with an idea that *an optimal kd-tree for computing q_i can be built incrementally.*

Ex-DPC computes the dependent point of each point in P in the following way:

(1) Destroy $\mathcal{K}$ (i.e., $\mathcal{K}$ becomes an empty set).
(2) Sort P in descending order of local density.
(3) Pick the front point of P, say p, set ∞ as its dependent distance, insert p into $\mathcal{K}$, and pop p.
(4) Pick the front point of P, say p', conduct a nearest neighbor search with query point p' on $\mathcal{K}$, set the result as its dependent point, insert p' into $\mathcal{K}$, and pop p'.
(5) Repeat the above operation until P has no points.

It is important to note that, for the front point p_i of P, the kd-tree contains only points whose local densities are higher than ρ_i. (We assume that all points have different local densities, which is practically possible by adding a random value $\in (0, 1)$ to ρ_i.) Therefore, for p_i, its nearest neighbor search retrieves the correct dependent point. We now prove that our approach is efficient.

Lemma 2. *Ex-DPC needs $O(n(n^{1-1/d} + \rho_{avg}))$ time to compute the dependent points of all points in P, when most points in P have $\delta \leq d_{cut}$ (i.e., when we have $\alpha(n-1)(n^{1-1/d} + \rho_{avg}) > (1-\alpha)(n-1)n$ for $\alpha \in [0, 1]$).*

Proof. Destroying $\mathcal{K}$ and sorting P incur $O(n \log n)$ time, and inserting a point into an empty $\mathcal{K}$ incurs $O(1)$ time. Next, for a point $p \in P$, computing its dependent point incurs $O(n^{1-1/d} + \rho_i)$ time in the worst case if $\delta_i \leq d_{cut}$, which is seen from the range search result. Inserting p_i into $\mathcal{K}$ incurs $O(\log n)$ time. Therefore, the fourth operation needs $O(n^{1-1/d} + \rho_i)$ time if $\delta_i \leq d_{cut}$ (otherwise, it needs $O(n)$ time). Let α be the ratio that a given $p \in P$ has $\delta \leq d_{cut}$, i.e., αn points have this case. Since we repeat the above operation $(n-1)$ times, our approach incurs $O(\alpha(n-1)(n^{1-1/d} + \rho_{avg}) + (1-\alpha)(n-1)n)$ time. When most points in P have $\delta \leq d_{cut}$, i.e., α is sufficiently large, $\max\{\alpha(n-1)(n^{1-1/d} + \rho_{avg}), (1-\alpha)(n-1)n\} = \alpha(n-1)(n^{1-1/d} + \rho_{avg})$. Therefore, this lemma holds. □

In practice, most points $\in P$ have $\delta \leq d_{cut}$, thus this assumption is not strict.

Analysis. From Lemmas 1 and 2 and the time complexity of the label propagation, we have:

Theorem 1 (Time complexity of Ex-DPC). *The time complexity of Ex-DPC is $O(n(n^{1-1/d} + \rho_{avg}))$, when most points in P have $\delta \leq d_{cut}$.*

Now we see that, for d_{cut} yielding $n \cdot \rho_{avg} = o(n^2)$, Ex-DPC always needs time less than $O(n^2)$, and an arbitrary sufficiently small d_{cut}

[4] Ex-DPC (or our algorithms) can in fact employ a similar data structure, e.g., R-tree [28]. However, we do not consider the R-tree for our solution in this paper because it does not provide any theoretical performance guarantee.

satisfies it[5]. The space complexity of Ex-DPC is $O(n)$, as Ex-DPC employs a single kd-tree, whose space complexity is $O(n)$.

Implementation for parallel processing. We can parallelize the local density computation in Ex-DPC, but, unfortunately, its dependent point computation cannot be parallelized. This is derived from the fact that Ex-DPC needs to compute the dependent point of each point *one by one*, since the kd-tree is incrementally updated. Hence we focus on how to parallelize its local density computation.

Given P and multiple threads (or CPU cores), we parallelize local density computation by assigning each point in P to one of the threads. Then, each thread independently conducts a range search for each assigned point. To exploit the parallel processing environment (i.e., to hold load balancing), each thread should have (almost) the same processing cost. Recall that the range search cost of p_i is $O(n^{1-1/d} + \rho_i)$, indicating that the cost depends on its local density, which cannot be pre-known and is different between points. This means that a simple hash-partitioning of P may not hold load balancing. We therefore employ a heuristic that assigns a point to a thread *dynamically*. Specifically, for each thread, Ex-DPC assigns a point, and when a thread has finished its range search, Ex-DPC assigns another point to the thread. We use OpenMP for multi-threading, and to implement the above approach, "#pragma omp parallel for schedule (dynamic)" is used.

4 APPROX-DPC

Ex-DPC still has some weaknesses. (i) Ex-DPC incurs unnecessary kd-tree traversal, because points with small distances traverse almost the same nodes of $\mathcal{K}$. (ii) Ex-DPC has the hardness of parallelizing its dependent point computation. Our first approximation DPC algorithm, Approx-DPC, removes the above limitations to accelerate processing efficiency by joint range search and approximating dependent points. We prove that Approx-DPC provides the same cluster centers, given the same ρ_{min} and δ_{min} for Ex-DPC, rendering an accurate clustering result.

4.1 Data Structure

Approx-DPC also uses a kd-tree $\mathcal{K}$ to index P. In addition, Approx-DPC leverages another data structure, which is a uniform grid G, a set of non-empty cells, where each cell is a d-dimensional square with side length $\frac{d_{cut}}{\sqrt{d}}$ for all dimensions. Each cell c of the grid G maintains:

- $P(c)$: a set of points covered by c,
- $p^*(c)$: the point with the maximum local density among $P(c)$,
- $\min_{P(c)} \rho$: the minimum local density in $P(c)$, and
- $N(c)$: an identifier set of cells to which points $p \notin P(c)$ satisfying $dist(p^*(c), p) < d_{cut}$ belong.

An example of the above grid structure is illustrated in Figure 3.

Approx-DPC builds the grid G online, as it depends on d_{cut}. Given P, we sequentially access each point $p \in P$ and map it to its corresponding cell. If the cell has not been created, we create it before p is mapped, thereby no empty-cell is created. The information maintained in each cell is obtained during the local density computation.

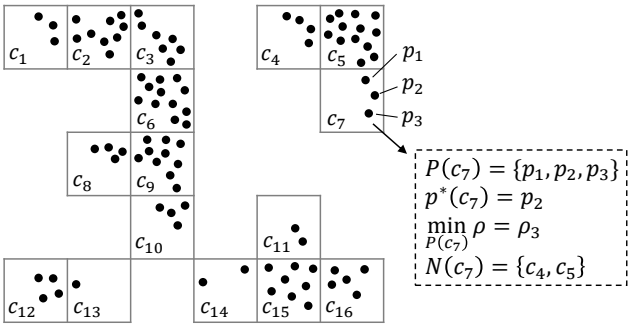


Figure 3: The grid structure in Approx-DPC. The black points represent points in P, and there are 16 cells in G for a given d_{cut}. The dashed box shows each element maintained in cell c_7 where ρ_2 is the largest among $P(c_7) = \{p_1, p_2, p_3\}$.

Some existing clustering studies, e.g., [17, 32], also employ grid-based data structures, but the details are totally different. It is important to note that our main contributions in §4 are new ideas (joint range search and cell-based dependent point approximation) which respectively derive efficient local density and dependent point computation, through our grid G.

4.2 Local Density Computation

Let $B(p_i, d_{cut})$ be the (open) ball centered at p_i with radius d_{cut}. If $dist(p_i, p_j)$ is small, $B(p_i, d_{cut})$ and $B(p_j, d_{cut})$ have a significant overlap. For instance, as can be seen in Figure 3, points in the same cell have this observation. This suggests that Ex-DPC incurs unnecessary kd-tree traversal (recall that Ex-DPC iteratively conducts a range search). Approx-DPC improves the performance of local density computation by using an idea: *if we can jointly search the points covered by each ball $B(p, d_{cut})$, where $p \in P(c)$, in a cell c, we can reduce unnecessary tree traversal.* For $p \in P$, Approx-DPC computes its exact local density, to guarantee that the same cluster centers are selected as those in Ex-DPC.

Joint range search. Given a cell c_i of G, we obtain points covered by each ball of a point $p \in P(c_i)$, through a single range search on $\mathcal{K}$. Let cp_i be the center point of c_i (i.e., the coordinates of cp_i are the center of c_i). Furthermore, define $p' = \arg\max_{p \in P(c_i)} dist(cp_i, p)$. It is important to observe that, for each $p \in P(c_i)$, we have $B(p, d_{cut}) \subseteq B(cp_i, d_{cut} + dist(cp_i, p'))$. This corresponds to the following: let $R(p, d_{cut})$ be the result of the range search with query point p and radius d_{cut}. Then $R(cp_i, d_{cut} + dist(cp_i, p'))$ is a superset of $R(p, d_{cut})$ for each $p \in P(c_i)$. Figure 4 depicts this observation, where the red point is the center point of cell c_7.

We proceed to present the procedure of the joint range search in a cell. Given a cell c_i of G, Approx-DPC

(1) makes cp_i (the center point of c_i),
(2) computes $\max_{p \in P(c_i)} dist(cp_i, p)$ (assume that $p' \in P(c_i)$ holds this),

[5] Actually, as long as $\rho_{avg} \leq n^{1-1/d}$, Ex-DPC is clearly sub-quadratic to n. As we assume small d_{cut}, this holds (for Ex-DPC, Approx-DPC, and S-Approx-DPC) in practice.

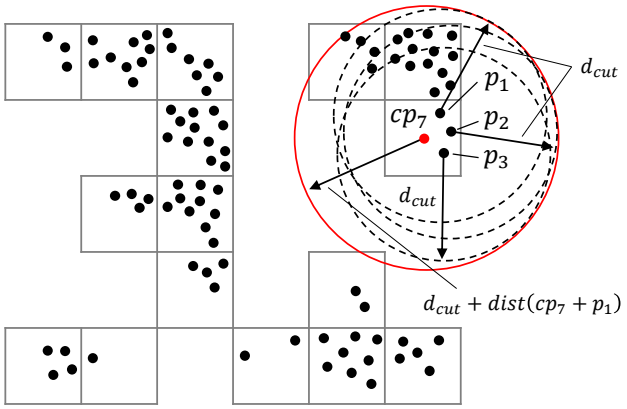


Figure 4: An example of joint range search

(3) obtains $R(cp_i, d_{cut} + dist(cp_i, p'))$ through a range search with query point cp_i and radius $d_{cut} + dist(cp_i, p')$ on $\mathcal{K}$, and
(4) for each $p_j \in P(c_i)$, scans $R(cp_i, d_{cut} + dist(cp_i, p'))$ to compute its exact local density ρ_j.

Algorithm. Given a cell c_i of G, Approx-DPC conducts a joint range search. During this, Approx-DPC computes the point with the maximum local density in c_i, $p^*(c_i)$, and the minimum local density in c_i, $\min_{P(c_i)} \rho$. When $p^*(c_i)$ is identified, $N(c_i)$, the identifier set of cells c_j that cover $p \notin P(c_i)$ such that $dist(p^*(c_i), p) < d_{cut}$, is obtained. Approx-DPC repeats this operation for every cell of G.

Time complexity. Let $\rho(cp_i) = |R(cp_i, d_{cut} + dist(cp_i, p'))|$.

Lemma 3. *Approx-DPC needs $O(\sum_G (n^{1-1/d} + \rho(cp_i) \cdot |P(c_i)|))$ time to compute the local densities of all points.*

Proof. Obviously, the main cost of the local density computation in Approx-DPC is derived from joint range searches, because the cell update (i.e., computing $p^*(c_i)$, $\min_{P(c_i)} \rho$, and $N(c_i)$) is done during the joint range search. The joint range search in a cell c_i consists of (i) the range search with query point cp_i and radius $d_{cut} + dist(cp_i, p')$ on the kd-tree and (ii) exact local density computation for each $p \in P(c_i)$ by scanning the range search result $R(cp_i, d_{cut} + dist(cp_i, p'))$. The cost of (i) is $O(n^{1-1/d} + \rho(cp_i))$, and the cost of (ii) is $\Theta(\rho(cp_i) \cdot |P(c_i)|)$. The joint range search time in c_i is then $O(n^{1-1/d} + \rho(cp_i) \cdot |P(c_i)|)$. Consequently, we have the lemma. □

Remark 1. *The local density computation in Approx-DPC beats the one in Ex-DPC in practice. For small d_{cut}, we have $\rho(cp_i) = O(\rho_j)$ in practice, where $p_j \in P(c_i)$. Let $|P(c_i)| = n_i$. We then have*

$$O(\sum_G (n^{1-1/d} + \rho(cp_i) \cdot n_i)) = O(\sum_G n^{1-1/d} + \sum_G \rho_j \cdot n_i) = O(\sum_G n^{1-1/d} + \sum_P \rho_j).$$

Because $|G| \leq n$, $\sum_G n^{1-1/d} \leq n^{2-1/d}$. Moreover, $\sum_P \rho_j = n \cdot \rho_{avg}$. We now see the improvement of local density computation against the one in Ex-DPC.

4.3 Dependent Point Computation

Ex-DPC demonstrates that, for each point in P, its dependent point is obtained efficiently, but parallelizing it is hard. A challenge here is how to compute dependent points while holding efficiency and parallelizability. We solve this challenge by allowing *approximate dependent points*, and design a fast algorithm based on the following key ideas: (i) Clustering accuracy would not degrade as long as

- an approximate dependent point of each point p_i is close to p_i, and
- we compute the exact dependent point for p_i if there are no close points with higher local density than ρ_i.

(ii) It is easy to find an approximate dependent point of p_i if we use the information maintained in each cell of G.

Approximate computation. To implement the above key ideas, Approx-DPC allows the following: *for a point $p_i \in P$, if there exists a point p_j such that $\rho_i < \rho_j$ and $dist(p_i, p_j) \leq d_{cut}$, p_j can be an approximate dependent point of p_i*. Due to this, many points in P can have their approximate dependent points in a constant time. Specifically, Approx-DPC computes an approximate dependent point of p_i, which belongs to a cell c, based on the following rules:

- If $p_i \neq p^*(c)$, its approximate dependent point is $p^*(c)$, (The distance to each point in the same cell is at most d_{cut}.) The dependent distance of p_i is set as d_{cut}, instead of computing the distance between p_i and its approximate dependent point. (We later show that it does not matter.)
- If $p_i = p^*(c)$, Approx-DPC retrieves a cell c' from $N(c)$ such that $\min_{P(c')} \rho > \rho_i$. If there exists such a cell c', the approximate dependent point and dependent distance of p_i are $p^*(c')$ and d_{cut}, respectively. Otherwise, we do not decide its approximate dependent point here.

It is important to note that $|N(c)| = O(1)$ for an arbitrary fixed d, meaning that the above approach takes only $O(1)$ time for each point $p \in P$.

Exact computation. For a point p_i whose dependent point has not been decided in the above computation, Approx-DPC computes its exact dependent point. We take a different approach from Ex-DPC, and our approach in Approx-DPC still runs in $o(n^2)$ time and holds parallelizability. Our idea here is as follows: For p_i, we avoid accessing points with less local density than ρ by dividing P into some subsets and retrieve q_i by pruning unnecessary subsets.

First, we sort P in ascending order of local density. Second, we equally divide P into s disjoint subsets, P_1, ..., P_s (i.e., $P_i \cap P_j = \emptyset$ for $i \neq j$, and $\bigcup_s P_i = P$), and build a kd-tree $\mathcal{K}_i$ for each subset P_i. Note that, for $p \in P_i$ and $p' \in P_j$ ($i < j$), we have $\rho < \rho'$. We set s so that

$$\frac{n}{s} = O((s-1)(\frac{n}{s})^{1-1/d}), \tag{2}$$

to provide a theoretical performance guarantee (see Lemma 4). Let P' be the set of points whose dependent points have not been decided. Given a point $p_i \in P'$ and a subset P_j, as can be seen from Figure 5, we have three cases:

(i) All points in P_j have higher local density than ρ_i. In this case, we conduct a nearest neighbor search on $\mathcal{K}_j$.
(ii) P_j has not only points with higher local density than ρ_i but also points p_k that have $\rho_i \geq \rho_k$. In this case, we scan the

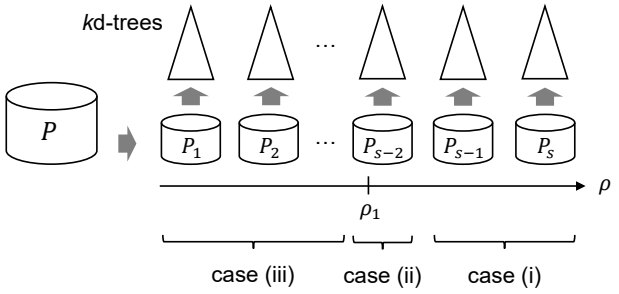


Figure 5: Partitioning P into s disjoint subsets, and the relationship between a point p_1 and the subsets

whole P_j and obtain the nearest neighbor point with higher local density than ρ_i in P_j. Note that there is at most one subset which has this case for p_i.

(iii) The local densities of all points in P_j are less than (or equal to) ρ_i. In this case, p_i ignores P_j.

The exact dependent point of $p_i \in P'$ is obtained by evaluating each subset P_j based on the above approach. The dependent distance of p_i, δ_i, follows Definition 3.

Time complexity. We analyze the efficiency of the dependent point computation in Approx-DPC. Clearly, its main cost is derived from the exact dependent point computation, since the approximate computation requires only $O(1)$ time for each $p \in P$. We thus consider the time complexity of the exact computation.

Lemma 4. *The time complexity of the dependent point computation in Approx-DPC is* $O(ns(\frac{n}{s})^{1-1/d})$.

Proof. Sorting P needs $O(n \log n)$ time, and dividing P into s subsets needs $O(n)$ time. Building a kd-tree for each subset requires $O(s \cdot \frac{n}{s} \log \frac{n}{s}) = O(n \log \frac{n}{s})$. For a point $p_i \in P'$, its worst case is to conduct a nearest neighbor search on $\mathcal{K}_2$, ..., $\mathcal{K}_s$ and scan the whole P_1. This case incurs $O((s-1)(\frac{n}{s})^{1-1/d})$ from Equation (2). Since $|P'| = O(n)$, we have the lemma. □

Equation (2) implies that s never reaches n (and s is small in practice), so this lemma theoretically demonstrates the efficiency of the dependent point computation in Approx-DPC. It is important to note that, since many points obtain their approximate dependent points in $O(1)$ time, meaning $|P'| \ll n$, its practical cost is much less than the one in Lemma 4. In addition, we will clarify that this approach is parallel-friendly in §4.5.

4.4 Analysis

Next, we consider the overall performance of Approx-DPC. Lemmas 3 and 4 and the fact that $|P'| \ll n$ prove the following theorem.

Theorem 2 (Time complexity of Approx-DPC). *Approx-DPC requires* $O(\sum_G (n^{1-1/d} + \rho(cp_i) \cdot |P(c_i)|))$ *for an arbitrary fixed* d.

Remark 1 has already claimed that this time complexity is better than that of Ex-DPC in practice. Besides, Approx-DPC has reasonable space complexity.

Theorem 3 (Space complexity of Approx-DPC). *The space complexity of Approx-DPC is* $O(n)$ *for an arbitrary fixed* d.

Proof. Approx-DPC employs a kd-tree and a grid. The kd-tree $\mathcal{K}$ requires $O(n)$ space, and $\bigcup P(c) = P$. In addition, $|N(c)| = O(1)$ for an arbitrary fixed d. The exact dependent point computation in Approx-DPC builds s kd-trees, each of which has $O(\frac{n}{s})$ space, i.e., $O(n)$ in total. This concludes that the theorem is true. □

Why accurate? Approx-DPC provides a highly accurate clustering result, which is empirically demonstrated in §6. We discuss why we have this. To start with, it is important to see:

Theorem 4 (Cluster center guarantee). *Approx-DPC provides the same cluster centers as Ex-DPC, given the same* δ_{min} *and* ρ_{min}.

Proof. For a point $p_i \in P$, Approx-DPC approximates its dependent distance, which is d_{cut}, iff there exists a point p_j such that $\rho_i < \rho_j$ and $dist(p_i, p_j) \leq d_{cut}$. Recall that Approx-DPC computes the exact local density for all points in P and finds the nearest neighbor point with higher local density for points $p_i \in P$ that do not have points p_j satisfying $\rho_i < \rho_j$ and $dist(p_i, p_j) \leq d_{cut}$. This leads to that (1) noises are selected correctly and (2) Approx-DPC computes the *exact dependent distance* for points p_i with $\delta_i > d_{cut}$. Since $\delta_{min} > d_{cut}$, this theorem holds. □

Assuming that there is a link (or an edge) between p_i and p_j, where p_j is the dependent point of p_i, a cluster in DPC is considered to be a tree rooted at a cluster center [21]. In this sense, a cell c_i is a sub-tree rooted at $p^*(c_i)$. Also, points whose dependent distances are exactly computed (i.e., the roots of some sub-trees) are the stem of the tree (i.e., cluster). The other sub-trees, which have approximate dependent points, are then considered to be branches connecting to (some parts of) the stem. It is intuitively seen that, as long as the stem is exact and sub-trees are connected to it, the cluster is accurate. Approx-DPC yields the exact stem and makes the other sub-trees connect to some part of the stem, thereby it forms an accurate cluster set. One exception is border points that exist around the border between different clusters. We study their influence in §6.

4.5 Parallelization

We parallelize Approx-DPC through a cost-based partitioning approach. In a nutshell, given multiple threads, Approx-DPC (i) estimates the cost of each task in local density computation and dependent point computation and (ii) assigns the task to a thread, so that each thread has almost the same sum cost, for load balancing. Although minimizing the sum cost difference between threads is NP-complete [2], we can have a good partitioning result in practice through a 3/2-approximation greedy algorithm [22]. This greedy algorithm takes $O(n't)$ time, where n' is the number of instances (cells or points) that are distributed to threads and t is the number of threads. This cost is trivial compared with those of the main operations, so we focus on how to estimate the cost of each task and how to parallelize Approx-DPC.

Parallel local density computation. The joint range search of a cell c_i consists of two main operations: obtaining $R(cp_i, d_{cut} + dist(cp_i, p'))$ and scanning it for each point in the cell. The real cost

of the first operation is hard to know in advance, since it depends on d_{cut} and the density around cp_i. However, we can estimate the density around cp_i with a trivial cost. Let $cost_{range}(c_i)$ be the estimated cost of the range search with query point cp_i and radius $d_{cut} + dist(cp_i, p')$, and

$$cost_{range}(c_i) = |P(c_i)|.$$

This is reasonable, because it clearly represents the density of the cell (if $|P(c_i)|$ is large/small, $|R(cp_i, d_{cut} + dist(cp_i, p'))|$ should be large/small). In addition, this estimation incurs a trivial computation cost, i.e., $O(1)$ time. Next, let $cost_{scan}(c_i)$ be the estimated cost of scanning $R(cp_i, d_{cut} + dist(cp_i, p'))$ for each point in c_i. We have

$$cost_{scan}(c_i) = |P(c_i)| \cdot |R(cp_i, d_{cut} + dist(cp_i, p'))|.$$

This estimation is possible in $O(1)$ time, after we obtain $R(cp_i, d_{cut} + dist(cp_i, p'))$. We therefore employ a 2-phase approach.

We describe how to parallelize the local density computation in Approx-DPC. Approx-DPC computes $cost_{range}(c_i)$ for each cell c_i in the grid. Given multiple threads, Approx-DPC assigns c_i to a thread, based on $cost_{range}(c_i)$ and the greedy algorithm. Each thread then independently conducts the range search and obtains $R(cp_i, d_{cut} + dist(cp_i, p'))$ for each assigned cell c_i. Approx-DPC next estimates $cost_{scan}(c_i)$ for each cell c_i, and again assigns c_i to a thread, based on the same approach as the previous assignment. Then, each thread independently computes the exact local densities of points in the assigned cells.

Parallel dependent point computation. As for a point $p \in P$, Approx-DPC computes an approximate dependent point iff there exists a point that satisfies the approximation rules. This is independent of computing approximate dependent points of the other points, so an approximate dependent point of each point can be computed in parallel. Similarly, sorting P, dividing P into s subsets, and building a kd-tree for each subset can be also simply parallelized by assigning each point in P or each subset P_i into a given thread. We hence consider the cost of computing the exact dependent point of $p \in P'$ (P' is the set of points whose dependent points are exactly computed).

Now recall that its cost is dependent on the number of subsets P_i which may have the dependent point of p. Let m be the number, and we estimate the cost as follows:

$$cost_{dep}(p) = \begin{cases} \frac{n}{s} + (m-1)(\frac{n}{s})^{1-1/d} & \text{(if } p \text{ has case (ii))} \\ m(\frac{n}{s})^{1-1/d} & \text{(otherwise),} \end{cases}$$

where $cost_{dep}(p)$ is an estimated cost. We parallelize the exact dependent point computation by assigning $p \in P'$ to a thread based on $cost_{dep}(p)$ and the greedy algorithm.

5 S-APPROX-DPC

Approx-DPC improves the performance of DPC against Ex-DPC without any additional parameters. However, if applications allow a rough clustering result but require to obtain it more quickly, another solution is needed. To this end, we propose S-Approx-DPC, which incorporates the following additional idea.

Main idea. Given $P' = \{p_i, ...\} \subset P$, where points in P' are close to each other, it is important to observe that points in P' have almost the same local density. Therefore their dependent points are the same or exist in a close area. From this observation, we can consider that the clustering result does not change much, even if we pick only one point, say p_i, from P', set p_i as the (approximate) dependent point of the other points p_j in P', and do nothing for p_j. (For S-Approx-DPC, ρ_{min} is not applicable to p_j.) Conceptually, S-Approx-DPC reduces the processing time by converting point clustering into cell clustering. This reduces the number of range searches and the time to retrieve dependent points.

Data structure. S-Approx-DPC also assumes that P is indexed by a kd-tree, and builds a grid G' online. Each cell of G' is a d-dimensional square with side length $\frac{\epsilon \cdot d_{cut}}{\sqrt{d}}$, where ϵ (> 0) is a user-specified approximation parameter. Hereafter, we use c to denote a cell of G'. Different from the cells in Approx-DPC, each cell c of G' does not maintain $p^*(c)$ and $\min_{P(c)} \rho$, and $N(c)$ is defined as an identifier set of cells to which points $p' \notin P(c)$ satisfying $dist(p, p') < d_{cut}$, where p is a sampled point from $P(c)$ belong. (We can deterministically decide p in an arbitrary way.)

Local density computation. S-Approx-DPC takes a similar approach to Ex-DPC. For each cell of G', S-Approx-DPC picks one point, say p, from $P(c)$, conducts a range search with query point p and radius d_{cut} on the kd-tree, and obtains $N(c)$.

From Lemma 1, we have:

Corollary 1. *The time complexity of the local density computation in S-Approx-DPC is* $O(\sum_{G'}(n^{1-1/d} + \rho_{avg}))$.

Dependent point computation. For each cell of G', S-Approx-DPC has points that have not been picked during the local density computation. S-Approx-DPC sets the picked point in the cell as their approximate dependent point. As for piked points, we retrieve their approximate dependent points by a similar approach to that of Approx-DPC. If there are some picked points p_i that cannot find their approximate dependent points in the approach, we form temporary clusters. We then find approximate dependent points while pruning temporary clusters that cannot have points with a small distance to p_i and higher local density than ρ_i. Below, we explain how to determine an approximate dependent point of a picked point.

- First phase. Let P_{pick} be a set of picked points. For the picked point p_i in a cell c, if there exist some picked points in cells $\in N(c)$ with higher local density than ρ_i, an approximate dependent point of p_i can be arbitrarily chosen from them. In this case, its approximate dependent distance is bounded by $(1+\epsilon)d_{cut}$, from the definition of G' and the fact that the distance between points in the same cell is at most $\epsilon \cdot d_{cut}$. (This bound is useful for determining δ_{min}.)

- Second phase. After the first phase, we have some picked points that do not have other picked points with higher local densities within $(1+\epsilon)d_{cut}$. Let P'_{pick} be a set of these points, and the remaining task is to retrieve an approximate dependent point of each point in P'_{pick}. For each point $p_i \in P'_{pick}$, S-Approx-DPC computes q'_i, its nearest point with higher local density than ρ_i among P_{pick}.

We here assume that $|P'_{pick}|^2 \leq O(n)$. (Otherwise, S-Approx-DPC employs the same approach as the exact dependent point computation in Approx-DPC.) The detail is as follows:

(1) S-Approx-DPC forms *temporary* clusters whose cluster centers are points in P'_{pick}, based on the dependency relationships computed in the first phase.
(2) Let T_i be a temporary cluster whose cluster center is p_i, that is, T_i is a set of picked points that are reachable from p_i. S-Approx-DPC computes $r_i = \max_{p \in T_i} dist(p_i, p)$.
(3) S-Approx-DPC next computes $p' = \arg\min dist(p_i, p_j)$, where $p_j \in P'_{pick}$ and $\rho_i < \rho_j$, for each $p_i \in P'_{pick}$.
(4) Then, for $p_i \in P'_{pick}$, a temporary cluster T_j cannot have q'_i if $dist(p_i, p_j) - r_j > dist(p_i, p')$ or $\rho_i \geq \rho_j$, from triangle inequality and definition. Therefore, S-Approx-DPC retrieves q'_i from the temporary clusters T_k such that $dist(p_i, p_k) - r_k \leq dist(p_i, p')$ and $\rho_i < \rho_k$.

This approach employs triangle inequality as with [7], but utilizes it in a more effective way, because our approach exploits intermediate clusters in DPC but [7] uses k-means clustering, whose clusters are totally different from density-based clusters. In addition, S-Approx-DPC has a theoretical efficiency:

Lemma 5. *The time complexity of the dependent point computation in S-Approx-DPC is $O(\sqrt{n}|G'|)$ for an arbitrary fixed d, if $|P'_{pick}|^2 \leq O(n)$.*

Proof. The first phase takes $O(|G'|)$ time for arbitrary fixed d, since $|N(c)| = O(1)$. In the second phase, forming temporary clusters needs $O(|G'|)$ time. Computing r_i for each T_i needs $O(\sum |T_i|) = O(|G'|)$ time, while we need $O(|P'_{pick}|^2) = O(n)$ time for computing p' for each $p \in P'_{pick}$. Last, computing q'_i requires at most $O(|G'|)$ if it cannot prune any temporary clusters, thereby the worst case incurs $O(|P'_{pick}||G'|)$ time. From $O(|P'_{pick}|) \leq O(\sqrt{n})$, the lemma holds. □

Analysis. From Corollary 1 and Lemma 5, we have:

Theorem 5 (Time complexity of S-Approx-DPC). *The time complexity of S-Approx-DPC is $O(\sum_{G'}(n^{1-1/d} + \rho_{avg}))$ for an arbitrary fixed d.*

We here discuss that the time complexity of S-Approx-DPC can be almost linear to n for fixed parameters, if the distribution of P does not change much and $|P'_{pick}|^2 \leq O(n)$. This is the main property of S-Approx-DPC. Fix d_{cut}, ϵ, and d. If the distribution of P does not change, $|G'|$ can be considered as a constant. This is because the distribution of cells in G' simply follows the distribution of P, thereby new cells are rarely created, even if n increases. Then $O(\sum_{G'}(n^{1-1/d} + \rho_{avg})) \approx O(n^{1-1/d} + \rho_{avg})$. The assumption that the distribution of P does not change also yields that ρ_{avg} grows linearly to n, resulting in the linear scalability of S-Approx-DPC to n for fixed parameters.

Notice that, if ϵ becomes larger, $|G'|$ becomes smaller. This leads to that the computation time is reduced but clustering accuracy may decrease. Although S-Approx-DPC does not guarantee a trade-off relationship between efficiency and accuracy, S-Approx-DPC tends to have this relationship empirically.

Last, from essentially the same proof of Theorem 3,

Corollary 2 (Space complexity of S-Approx-DPC). *The space complexity of S-Approx-DPC is $O(n)$ for an arbitrary fixed d.*

Implementation for parallel processing. For local density computation, S-Approx-DPC takes the same approach as Ex-DPC. Similarly to the local density computation, the operations in the dependent point computation of S-Approx-DPC consist of iterations, which can be simply parallelized by hash-partitioning or the same approach as the parallel local density computation. That is, S-Approx-DPC is also parallel-friendly.

6 EXPERIMENTS

Our experiments were conducted on a machine with dual 12-core Intel Xeon E5-2687W v4 processors (3.0GHz) that share a 512GB RAM. By using hyper-threading, this machine can run at most 48 ($= 2 \times 12 \times 2$) threads. All evaluated algorithms were implemented in C++, and we used OpenMP for multi-threading.

Datasets. We report the results on five synthetic dataset (*Syn*, *S1*, *S2*, *S3*, and *S4*) and four real datasets (*Airline*, *Household*, *PAMAP2*, and *Sensor*). The synthetic datasets are used for effectiveness evaluation. Syn is a 2-dimensional dataset with 100,000 points, which was generated based on a random walk model introduced in [17]. The domain of each dimension in Syn is $[0, 10^5]$. Airline[6] is a 3-dimensional dataset with 5,810,462 points, and its domain of each dimension is $[0, 10^6]$. Household and PAMAP2 are 4-dimensional datasets with respectively 2,049,280 and 3,850,505 points. Last, Sensor is an 8-dimensional dataset with 928,991 points. The domain of each dimension in Household, PAMAP2, and Sensor is the same as that of Syn, and they are from UCI machine learning archive[7].

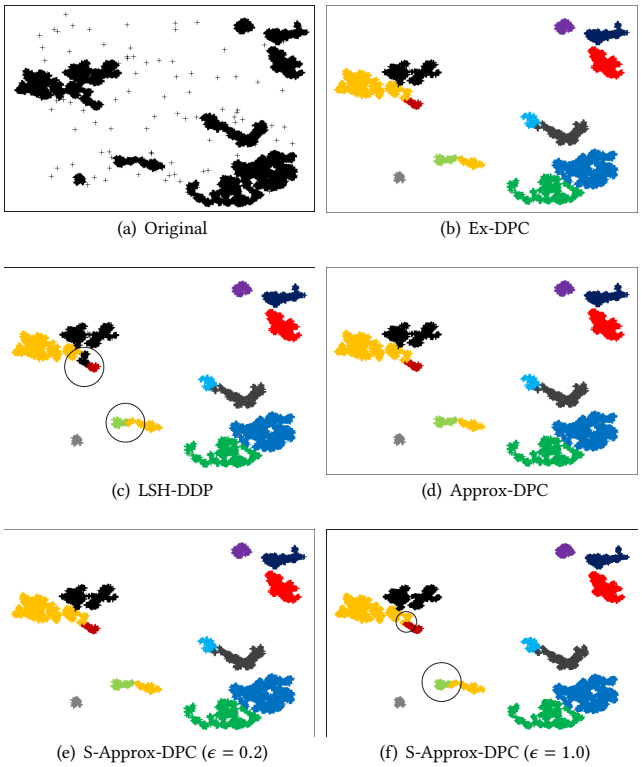
Algorithms. Our experiments evaluated the following algorithms:

- *Scan*: the straightforward algorithm introduced in §2.1.
- *R-tree + Scan*: a variant of Scan which computes local density by using an in-memory R-tree.
- *LSH-DDP* [42]: a state-of-the-art approximation algorithm.
- *CFSFDP-A* [7]: a state-of-the-art exact algorithm.
- *Ex-DPC*: our exact algorithm introduced in §3.
- *Approx-DPC*: our approximation algorithm introduced in §4.
- *S-Approx-DPC*: our approximation algorithm introduced in §5.

Codes are available in a GitHub repository[8]. We followed the original paper to set the inner parameters of LSH-DDP and CFSFDP-A. Table 1 shows that the dependent distance computation of CFSFDP-A is slower than that of Scan. We hence used the approach of Scan for computing dependent distances in CFSFDP-A.

Other approximation algorithms, FastDPeak [10], DPCG [39], and CFSFDP-DE [7] were also tested. We confirmed that FastDPeak and DPCG are slow (e.g., FastDPeak and DPCG respectively took 8114 and 14390 seconds on Airline at the default parameter setting) and significantly outperformed by our *exact* algorithm. In addition, we observed that the clustering accuracy (Rand index) of CFSFDP-DE is quite low (e.g., 0.18 on PAMAP2). We therefore omit the detailed results of these algorithms to keep space limitation[9].

[6] http://stat-computing.org/dataexpo/2009/
[7] https://archive.ics.uci.edu/ml/index.php
[8] https://github.com/amgt-d1/DPC
[9] As existing works, e.g., [42], show that the output of DPC is different from those of the other clustering algorithms, we do not test them.



(a) Original (b) Ex-DPC (c) LSH-DDP (d) Approx-DPC (e) S-Approx-DPC ($\epsilon = 0.2$) (f) S-Approx-DPC ($\epsilon = 1.0$)

Figure 6: 2D visualization of the clustering result of each algorithm ($d_{cut} = 250$)

6.1 Effectiveness Evaluation

2D visualization. To visually understand the effectiveness of our approximation algorithms, we used Syn, which is depicted in Figure 6(a) and has 13 density-peaks. We set $d_{cut} = 250$. In addition, ρ_{min} is specified so that noises can be clearly removed, while δ_{min} is specified so that we have 13 clusters. The clustering result of Ex-DPC, which is shown in Figure 6(b) where points in each cluster are illustrated by the same color, is used as ground-truth.

Let us first focus on Figure 6(c), the clustering result of the state-of-the-art approximation algorithm LSH-DDP. It has two major differences from that of Ex-DPC (specified by circles). Since LSH-DDP approximates both local density and dependent point, for a point p_i, it may decide that an approximate dependent point of p_i is p_j even if we indeed have $\rho_i \geq \rho_j$. This makes it hard for analysts to know why a cluster (e.g., the black one in LSH-DDP) is different from the exact one. This is a drawback of LSH-DDP.

We look at Figure 6(d), the clustering result of Approx-DPC. Actually this is the same clustering result as that of Ex-DPC. This result demonstrates that our key idea (for $p \in P$, compute an approximate dependent point if there exists a close point with higher local density; otherwise, compute the exact one) is promising.

We turn our attention to the clustering result of S-Approx-DPC. Figures 6(e) and 6(f) respectively illustrate the cases where $\epsilon = 0.2$ and $\epsilon = 1.0$. The case where $\epsilon = 0.2$ also returns the correct clustering result. This is reasonable, because $\epsilon = 0.2$ creates many cells and each approximate dependent point tends to belong to the same cluster as the exact dependent point. On the other hand, the case where $\epsilon = 1.0$ has one major difference and one minor difference. If a point has an approximate dependent point, it may belong to a different cluster from the exact one. In S-Approx-DPC, the approximate dependent distance of each picked point is guaranteed to be larger than (or equal to) the exact dependent distance (recall that picked points have *exact* local densities). For a point p that exists at a border between different clusters, even a small distance difference influences its cluster label. For instance, the nearest neighbor point with higher local density than ρ belongs to cluster C_1, but a point close to p belongs to C_2. This observation derives the difference specified by the circles in Figure 6(f).

Table 2: Rand index of LSH-DDP, Approx-DPC, and S-Approx-DPC on Syn with different noise rate

Noise rate	LSH-DDP	Approx-DPC	S-Approx-DPC
0.01	0.999	**1.000**	0.995
0.02	0.980	**0.984**	0.980
0.04	0.979	**0.983**	**0.983**
0.08	0.981	**0.982**	**0.982**
0.16	0.969	**0.976**	0.970

Table 3: Rand index of LSH-DDP, Approx-DPC, and S-Approx-DPC on S1, S2, S3, and S4

Dataset	LSH-DDP	Approx-DPC	S-Approx-DPC
S1	0.996	**1.000**	0.999
S2	0.994	**0.998**	0.996
S3	0.989	**0.999**	0.988
S4	0.979	**0.990**	0.981

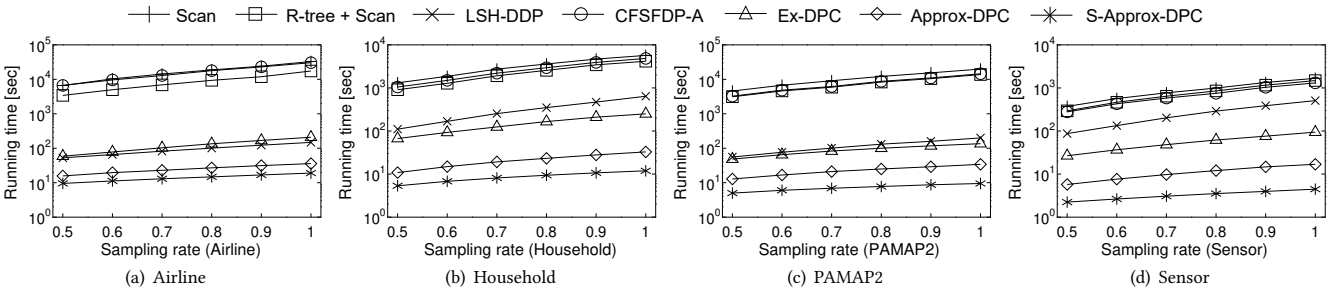
Table 4: Rand index of LSH-DDP and Approx-DPC on real datasets

	Airline	Household	PAMAP2	Sensor
LSH-DDP	0.938	0.983	0.951	0.902
Approx-DPC	**0.999**	**0.996**	**0.996**	**0.960**

Quantitative evaluation. We next investigate the accuracy of the approximation algorithms. We used Rand index to measure the accuracy of each approximation algorithm under the same parameter setting as Ex-DPC (i.e., the clustering result of Ex-DPC is the ground truth).

First, we investigate the robustness of the approximation algorithms to noise rate. We varied the noise rate of Syn, and Table 2 shows the result, where $\epsilon = 1.0$ for S-Approx-DPC. (Bold shows the winner.) From the result, we see that their accuracy is still high even when Syn has many noises (e.g., the rate is 0.16), showing the robustness to noises.

Furthermore, we investigate the robustness to the degree of cluster overlapping by using S1, S2, S3, and S4. These datasets have 15 Gaussian clusters and the same cardinality, whereas the degree of



(a) Airline (b) Household (c) PAMAP2 (d) Sensor

Figure 7: Impact of cardinality (sampling rate)

Table 5: Running time [sec] vs. accuracy (Rand index) of S-Approx-DPC

	Airline		Household	
ϵ	Time	Rand index	Time	Rand index
0.2	32.178	0.998	59.597	0.995
0.4	29.992	0.996	27.637	0.994
0.6	25.935	0.985	16.470	0.994
0.8	20.401	0.976	11.097	0.993
1.0	16.449	0.969	7.527	0.991

cluster overlapping of Sx increases as x increases, see [16]. That is, this experiment studies the impact of cluster overlapping rate. Table 3 shows that they provide almost perfect results even when the degree is high, i.e., on S4 (we confirmed that Ex-DPC and the approximation algorithms provide 15 clusters). This result demonstrates the robustness of our algorithms to cluster overlapping. (We confirmed that the noise rate and cluster overlapping degree do not affect the efficiencies of our algorithms.)

Table 4 shows the Rand index of LSH-DDP and Approx-DPC on real datasets. We set $d_{cut} = 1000$ for Airline, Household, and PAMAP2, and $d_{cut} = 5000$ for Sensor (these are default values of d_{cut}). For each dataset, we specified ρ_{min} and δ_{min} based on the discussion in Section 2. As with the 2D visualization, Approx-DPC provides a highly accurate clustering result and beats LSH-DDP.

We study the impact of ϵ on the Rand index of S-Approx-DPC, and the result is shown in Table 5 (we omit the results on PAMAP2 and Sensor, because they are similar to those on Airline and Household). The Rand index decreases with increase of ϵ. Because the cell size of G' becomes larger as ϵ increases, the approximate dependent point of each point can be more rough. As mentioned before, the approximate dependent distance of each picked point is larger than its exact dependent distance. For the same δ_{min} as Ex-DPC, therefore, S-Approx-DPC may provide more clusters, which also degrades the clustering accuracy. However, the impact of ϵ is small, i.e., the decrease in the Rand index is slight, and S-Approx-DPC beats LSH-DDP as well as Approx-DPC but its Rand index does not reach to that of Approx-DPC. Table 5 also shows the running time of S-Approx-DPC with 12 threads. From the relationship between running time and Rand index, we used 0.8, 0.8, 0.8, and 0.6 as ϵ for Airline, Household, PAMAP2, and Sensor, respectively.

6.2 Efficiency Evaluation

We report the running time of each algorithm. The default number of threads is 12.

Impact of cardinality. We first investigate the scalability of each algorithm to the number of points in a dataset. We varied the number of points in each dataset via uniform sampling, i.e., by varying sampling rate (the other parameter are fixed by their default values). Figure 7 plots the result.

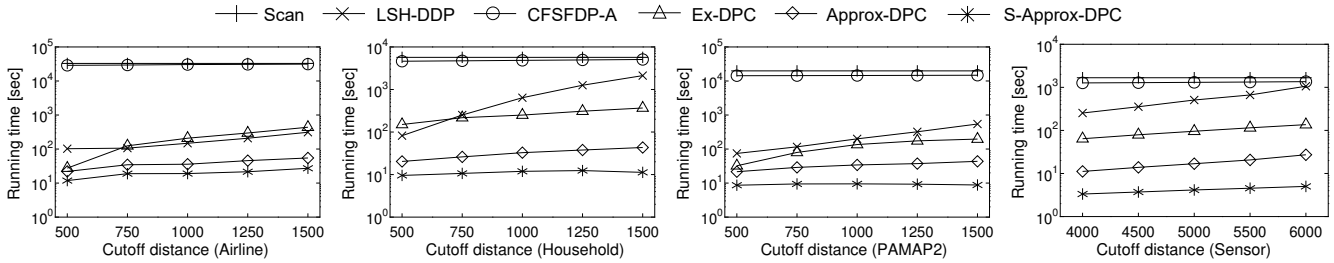
Let us focus on the exact algorithms (Scan, R-tree + Scan, CFSFDP-A, and Ex-DPC). Ex-DPC significantly outperforms the other exact algorithms. For example, when sampling rate is 1, the running time of Ex-DPC is 145.9, 19.3, 106.8, and 13.7 times faster than that of CFSFDP-A on Airline, Household, PAMAP2, and Sensor, respectively. Table 6 exhibits the time to compute local densities and dependent points of all points at the default parameter setting. (Since R-tree + Scan and CFSFDP-A employ the same dependent point computation as Scan, their δ comp. are blank but the same as that of Scan.) We see that Ex-DPC clearly improves both the computation, compared with the other exact algorithms, although Ex-DPC employs simple approaches. The R-tree alleviates the cost of local density computation, but R-tree + Scan still suffers from the quadratic cost of dependent point computation. CFSFDP-A also suffers from the quadratic time computation, and its running time (i.e., local density computation time) has a small gain against Scan, because its filtering technique is sensitive to noises. On the other hand, Ex-DPC scales much better, as it is sub-quadratic to n. Interestingly, Ex-DPC beats the existing approximation algorithm LSH-DDP on Household, PAMAP2, and Sensor. (We hereafter omit the result of R-tree + Scan, as its behavior is similar to that of Scan.)

Consider approximation algorithms. Approx-DPC outperforms the exact algorithms and LSH-DDP. When the sampling rate is 1, the running time of Approx-DPC is 4.1, 19.6, 5.8, and 30.1 times faster than that of LSH-DDP on Airline, Household, PAMAP2, and Sensor, respectively. Table 6 demonstrates that our joint range search improves the iterative range search (see ρ comp. of Ex-DPC and Approx-DPC). Besides, Approx-DPC scales better than LSH-DDP. S-Approx-DPC further improves the efficiency of DPC and actually scales linearly to sampling rate, as analyzed in §5.

Impact of d_{cut}. The main parameter of DPC is d_{cut}, thus we next study its impact. Figure 8 depicts the experimental result. The first observation is that Scan and CFSFDP-A are not sensitive to d_{cut}.

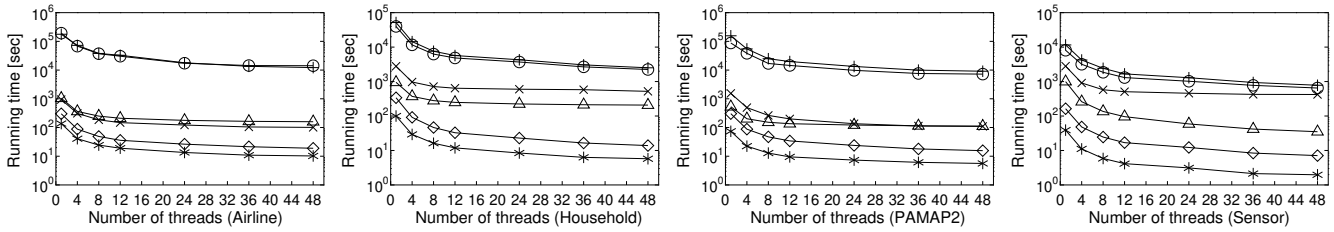
Table 6: Decomposed time [sec] (parameters are default ones)

	Airline		Household		PAMAP2		Sensor	
Algorithm	ρ comp.	δ comp.	ρ comp.	δ comp.	ρ comp.	δ comp.	ρ comp.	δ comp.
Scan	15492.70	17310.40	1703.37	3989.77	6114.82	13717.60	492.60	1178.27
R-tree + Scan	128.28	-	174.82	-	36.76	-	304.10	-
LSH-DDP	90.54	56.86	225.69	414.37	99.67	98.38	148.84	358.84
CFSFDP-A	13091.20	-	850.34	-	776.94	-	127.89	-
Ex-DPC	79.20	129.56	67.27	182.47	36.68	97.45	89.93	5.40
Approx-DPC	25.09	3.77	22.12	8.66	18.88	12.29	14.72	1.58
S-Approx-DPC	**11.24**	**1.16**	**7.92**	**0.74**	**6.08**	**0.72**	**3.56**	**0.27**



(a) Airline (b) Household (c) PAMAP2 (d) Sensor

Figure 8: Impact of d_{cut}



(a) Airline (b) Household (c) PAMAP2 (d) Sensor

Figure 9: Impact of number of threads

This is reasonable, because they compute local density and dependent point based on scanning P. Second, LSH-DDP is very sensitive to d_{cut}. This parameter affects the bucket size of locality-sensitive hashing, and some buckets have many points when d_{cut} is large. This observation can be understood from Table 1. Our algorithms are also affected by d_{cut}, since their time complexities have ρ_{avg}. That is, as d_{cut} becomes larger, their running time tends to become larger. However, S-Approx-DPC is less sensitive to d_{cut}. As d_{cut} becomes larger, the number of cells in its grid G' becomes smaller, i.e., the number of points that conduct a range search becomes smaller. That is, the running time of S-Approx-DPC is influenced by ρ_{avg} and $|G'|$.

Impact of number of threads. We investigate the scalability to the number of threads. Figure 9 shows that each algorithm normally improves its running time with increase of available threads. We see that Scan and CFSFDP-A are slow even when using 48 threads, and LSH-DDP is affected by the distribution of a given dataset. For example, LSH-DDP scales well to the number of threads on Airline and PAMAP2 but does not on the other datasets, because it does not consider load balancing when partitioning P.

The limitation of Ex-DPC (dependent point computation cannot be parallelized) is observed from the result. As we have more threads, the main overhead of Ex-DPC becomes dependent point computation, then its running time cannot be reduced. On Sensor dataset, Ex-DPC seems to scale well, because the main cost is derived from local density computation, as shown in Table 6. On the other hand, Figure 9 demonstrates that Approx-DPC and S-Approx-DPC can exploit available threads. For example, when using 48 threads, Approx-DPC terminates clustering within 20 seconds. Besides, in this case, Approx-DPC becomes 15.9, 24.4, 19.4, and 23.0 times faster, compared with the case of a single thread, on Airline, Household, PAMAP2, and Sensor, respectively. S-Approx-DPC has

Table 7: Memory usage [MB]

	Airline	Household	PAMAP2	Sensor
R-tree + Scan	564	346	277	133
LSH-DDP	2061	756	1455	342
CFSFDP-A	59362	12601	32206	3900
Ex-DPC	**461**	**171**	**321**	**93**
Approx-DPC	1316	422	790	201
S-Approx-DPC	1410	482	884	216

a similar result. (The reason why Approx-DPC and S-Approx-DPC cannot achieve 48 times faster is the hardware problem, i.e., two CPUs share a RAM and hyper-threading.)

Memory usage. Last, we study the memory usage of the evaluated algorithms by using default setting. Table 7 shows the result. Ex-DPC consumes almost the same memory as R-tree, and our approximation algorithms need less memory than LSH-DDP and CFSFDP-A. The memory usage of our approximation algorithms is higher than that of Ex-DPC, since they use a grid as an additional index. In addition, ϵ is less than 1 for S-Approx-DPC, so it creates more cells than Approx-DPC. Therefore, S-Approx-DPC needs more memory usage than Approx-DPC.

7 CONCLUSION

Density-based clustering is an important technique for many data mining tasks, and Density-Peaks Clustering is one of density-based clustering techniques and has many applications. However, its computational efficiency has not been considered much so far, although its straightforward implementation incurs $O(n^2)$ time, where n is the number of input points.

In this paper, we proposed efficient Density-Peaks Clustering algorithms, Ex-DPC, Approx-DPC, and S-Approx-DPC. As long as the average local density is sufficiently small, our algorithms are sub-quadratic. In addition, we considered multicore-based parallel processing, which is being popularized in many systems, and presented how to parallelize our algorithms. Because Ex-DPC cannot be fully parallelized, Approx-DPC and S-Approx-DPC are carefully designed so that they can parallelize main operations by allowing approximate results. Our experimental results have confirmed that Ex-DPC is much faster than the state-of-the-art exact algorithm, and Approx-DPC and S-Approx-DPC are more accurate, efficient, and scalable than the state-of-the-art approximation algorithms.

ACKNOWLEDGMENTS

This research is partially supported by JSPS Grant-in-Aid for Scientific Research (A) Grant Number 18H04095, JST CREST Grant Number J181401085, and JST PRESTO Grant Number JPMJPR1931.

REFERENCES

[1] Charu C. Aggarwal and Chandan K. Reddy (Eds.). 2014. *Data Clustering: Algorithms and Applications.*
[2] Daichi Amagata and Takahiro Hara. 2019. Identifying the Most Interactive Object in Spatial Databases. In *ICDE*. 1286–1297.
[3] Daichi Amagata and Takahiro Hara. 2021. Fast Density-Peaks Clustering: Multicore-based Parallelization Approach. In *SIGMOD*. 49–61.
[4] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering Points to Identify the Clustering Structure. In *ACM SIGMOD Record*. 49–60.
[5] David Arthur and Sergei Vassilvitskii. 2007. k-means++: The Advantages of Careful Seeding. In *SODA*. S, 1027–1035.
[6] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Scalable k-means++. *PVLDB* 5, 7 (2012), 622–633.
[7] Liang Bai, Xueqi Cheng, Jiye Liang, Huawei Shen, and Yike Guo. 2017. Fast Density Clustering Strategies based on the k-means Algorithm. *Pattern Recognition* 71 (2017), 375–386.
[8] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (1975), 509–517.
[9] Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. 2019. Solving k-center Clustering (with Outliers) in MapReduce and Streaming, Almost as Accurately as Sequentially. *PVLDB* 12, 7 (2019), 766–778.
[10] Yewang Chen, Xiaoliang Hu, Wentao Fan, Lianlian Shen, Zheng Zhang, Xin Liu, Jixiang Du, Haibo Li, Yi Chen, and Hailin Li. 2020. Fast Density Peak Clustering for Large Scale Data based on kNN. *Knowledge-Based Systems* 187 (2020), 104824.
[11] Yewang Chen, Shengyu Tang, Nizar Bouguila, Cheng Wang, Jixiang Du, and HaiLin Li. 2018. A Fast Clustering Algorithm based on Pruning Unnecessary Distance Computations in DBSCAN for High-dimensional Data. *Pattern Recognition* 83 (2018), 375–387.
[12] Zengjian Chen, Jiayi Liu, Yihe Deng, Kun He, and John E Hopcroft. 2019. Adaptive Wavelet Clustering for Highly Noisy Data. In *ICDE*. 328–337.
[13] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive Hashing Scheme based on p-stable Distributions. In *SoCG*. 253–262.
[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*. 226–231.
[15] Jose M Faleiro and Daniel J Abadi. 2017. Latch-free Synchronization in Database Systems: Silver Bullet or Fool's Gold?. In *CIDR*. 9.
[16] Pasi Fränti and Sami Sieranoja. 2018. K-means Properties on Six Clustering Benchmark Datasets. *Applied Intelligence* 48, 12 (2018), 4743–4759.
[17] Junhao Gan and Yufei Tao. 2015. DBSCAN Revisited: Mis-claim, Un-fixability, and Approximation. In *SIGMOD*. 519–530.
[18] Junhao Gan and Yufei Tao. 2017. Dynamic Density based Clustering. In *SIGMOD*. 1493–1507.
[19] Junhao Gan and Yufei Tao. 2017. On the Hardness and Approximation of Euclidean DBSCAN. *ACM Transactions on Database Systems* 42, 3 (2017), 14.
[20] Junhao Gan and Yufei Tao. 2018. Fast Euclidean OPTICS with Bounded Precision in Low Dimensional Space. In *SIGMOD*. 1067–1082.
[21] Shufeng Gong, Yanfeng Zhang, and Ge Yu. 2017. Clustering Stream Data by Exploring the Evolution of Density Mountain. *PVLDB* 11, 4 (2017), 393–405.
[22] Ronald L. Graham. 1969. Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.* 17, 2 (1969), 416–429.
[23] Shalmoli Gupta, Ravi Kumar, Kefu Lu, Benjamin Moseley, and Sergei Vassilvitskii. 2017. Local Search Methods for k-means with Outliers. *PVLDB* 10, 7 (2017), 757–768.
[24] Ruizhen Hu, Wenchao Li, Oliver Van Kaick, Hui Huang, Melinos Averkiou, Daniel Cohen-Or, and Hao Zhang. 2017. Co-locating Style-defining Elements on 3D Shapes. *ACM Transactions on Graphics* 36, 3 (2017), 33.
[25] Alessandro Lulli, Matteo Dell'Amico, Pietro Michiardi, and Laura Ricci. 2016. NG-DBSCAN: Scalable Density-based Clustering for Arbitrary Data. *PVLDB* 10, 3 (2016), 157–168.
[26] Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv preprint arXiv:1802.03426* (2018).
[27] Rashid Mehmood, Saeed El-Ashram, Rongfang Bie, Hussain Dawood, and Anton Kos. 2017. Clustering by fast search and merge of local density peaks for gene expression microarray data. *Scientific Reports* 7 (2017), 45602.
[28] Jianzhong Qi, Yufei Tao, Yanchuan Chang, and Rui Zhang. 2018. Theoretically Optimal and Empirically Efficient R-trees with Strong Parallelizability. *PVLDB* 11, 5 (2018), 621–634.
[29] Kun Ren, Jose M Faleiro, and Daniel J Abadi. 2016. Design Principles for Scaling Multi-core OLTP under High Contention. In *SIGMOD*. 1583–1598.
[30] Alex Rodriguez and Alessandro Laio. 2014. Clustering by Fast Search and Find of Density Peaks. *Science* 344, 6191 (2014), 1492–1496.
[31] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. 2017. DBSCAN Revisited, Revisited: Why and How You Should (still) use DBSCAN. *ACM Transactions on Database Systems* 42, 3 (2017), 19.
[32] Hwanjun Song and Jae-Gil Lee. 2018. RP-DBSCAN: A Superfast Parallel DBSCAN Algorithm based on Random Partitioning. In *SIGMOD*. 1173–1187.
[33] Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. 2017. *Handbook of Discrete and Computational Geometry.*
[34] Liudmila Ulanova, Nurjahan Begum, Mohammad Shokoohi-Yekta, and Eamonn Keogh. 2016. Clustering in the Face of Fast Changing Streams. In *SDM*. 1–9.
[35] Guangtao Wang and Qinbao Song. 2016. Automatic Clustering via Outward Statistical Testing on Density Metrics. *IEEE Transactions on Knowledge and Data*

Engineering 28, 8 (2016), 1971–1985.

[36] Wenguan Wang, Jianbing Shen, Fatih Porikli, and Ruigang Yang. 2018. Semi-supervised Video Object Segmentation with Super-trajectories. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41, 4 (2018), 985–998.

[37] Yiqiu Wang, Yan Gu, and Julian Shun. 2020. Theoretically-Efficient and Practical Parallel DBSCAN. In *SIGMOD*. 2555–2571.

[38] Zhen Wang, Rui Zhang, Jianzhong Qi, and Bo Yuan. 2019. DBSVEC: Density-Based Clustering Using Support Vector Expansion. In *ICDE*. 280–291.

[39] Xiao Xu, Shifei Ding, Mingjing Du, and Yu Xue. 2018. DPCG: An Efficient Density Peaks Clustering Algorithm based on Grid. *International Journal of Machine Learning and Cybernetics* 9, 5 (2018), 743–754.

[40] Keyu Yang, Yunjun Gao, Rui Ma, Lu Chen, Sai Wu, and Gang Chen. 2019. DBSCAN-MS: Distributed Density-Based Clustering in Metric Spaces. In *ICDE*. 1346–1357.

[41] Shuai Yang, Xipeng Shen, and Min Chi. 2019. Streamline Density Peak Clustering for Practical Adoptions. In *CIKM*. 49–58.

[42] Yanfeng Zhang, Shimin Chen, and Ge Yu. 2016. Efficient Distributed Density Peaks for Clustering Large Data Sets in MapReduce. *IEEE Transactions on Knowledge and Data Engineering* 28, 12 (2016), 3218–3230.

[43] Yang Zhang, Yunqing Xia, Yi Liu, and Wenmin Wang. 2015. Clustering Sentences with Density Peaks for Multi-document Summarization. In *NAACL-HLT*. 1262–1267.